**Exercise 2: Implementing the Factory Method Pattern - Documentation**

**Project Name:** FactoryMethodPatternExample

**Objective:**To implement a document creation system in Java that uses the Factory Method Pattern to generate different types of documents (Word, PDF, Excel) without tightly coupling the client code to specific classes.

**1. Introduction to Factory Method Pattern:**

The **Factory Method Pattern** is a creational design pattern that provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created.

**Scenario:**  
We need to create a document management system that supports different types of documents such as Word, PDF, and Excel. Using the Factory Method Pattern allows us to create these document types via specific factory classes while adhering to the Open/Closed Principle.

**2. Design Structure:**

* **Product Interface:** Document defines the interface for different document types.
* **Concrete Products:** WordDocument, PdfDocument, and ExcelDocument implement Document.
* **Creator (Abstract Factory):** DocumentFactory declares the factory method createDocument().
* **Concrete Factories:** WordFactory, PdfFactory, ExcelFactory implement the creation of specific document types.

**3. Implementation:**

**Document.java**

public interface Document {

void open();

}

**WordDocument.java**

public class WordDocument implements Document {

public void open() {

System.out.println("Opening a Word document...");

}

}

**PdfDocument.java**

public class PdfDocument implements Document {

public void open() {

System.out.println("Opening a PDF document...");

}

}

**ExcelDocument.java**

public class ExcelDocument implements Document {

public void open() {

System.out.println("Opening an Excel document...");

}

}

**DocumentFactory.java**

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordFactory.java**

public class WordFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

**PdfFactory.java**

public class PdfFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelFactory.java**

public class ExcelFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**TestFactory.java**

public class TestFactory {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = new PdfFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

DocumentFactory excelFactory = new ExcelFactory();

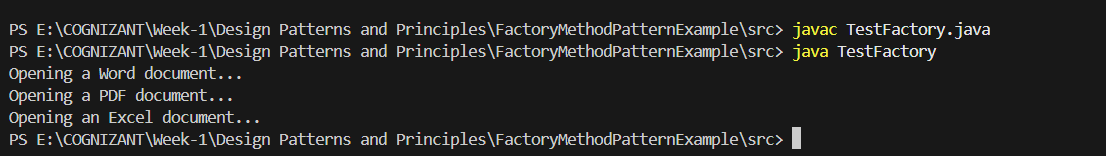
Document excel = excelFactory.createDocument();

excel.open();

}

}

**4. Output of the Program:**



**5. Design Decisions and Benefits:**

* **Decoupling:** Client code (TestFactory) doesn’t depend on concrete document classes.
* **Open/Closed Principle:** Easily extendable to new document types without modifying existing code.
* **Encapsulation of Creation Logic:** Each factory knows how to create its own document.

**6. Conclusion:**

This exercise demonstrates the Factory Method Pattern in a flexible and extensible manner. It allows the system to create objects without knowing their exact class, which improves maintainability, promotes reusability, and aligns with object-oriented design best practices.